Resource Description Framework (RDF)

Lecture 02

Resource Description Framework

- **Resource Description Framework (RDF)** is a formal language for describing structured information.
- Goal of RDF is to enable applications to exchange data on the Web while still preserving their original meaning.
- RDF consequently is often viewed as the basic representation format for developing the Semantic Web.

History of RDF

- First official RDF specification was published by the W3C in 1999.
- Initial focus was on representing **metadata** about Web resources (e.g., authorship, copyright)
- Concept of Semantic Web later expanded RDF to include broader semantic information representation.
- Thus, a revised and extended RDF specification published in 2004.

RDF at Present

- A wide range of practical tools are available today for working with RDF.
- Most programming languages provide libraries to read and write RDF documents.
- RDF stores (also known as triple stores) are used to store and manage large RDF datasets.
- Commercial database vendors are beginning to support RDF through specialized extensions.

RDF at Present

- RDF is widely used to exchange metadata in specific domains.
- Example applications:
  - **RSS 1.0**: Uses RDF for web news syndication.
  - **Adobe XMP**: Embeds RDF metadata in PDF files.
  - **SVG files**: Use RDF for annotations in vector graphics.

RDF Format

- An RDF document describes a **directed graph**, i.e. a set of **nodes** that are linked by **directed edges** (“arrows”).
- Both nodes and edges are labeled with identifiers to distinguish them.
- **E.g.** RDF graph describing the relationship between this book and the publisher.

RDF Prefers Graphs Over Tress

- XML encodes information using tree structures, which are well-suited for hierarchical data.
- Tree structures are ideal for organizing electronic documents with strict hierarchies.
- Information in trees can often be accessed and processed efficiently.
- Despite these advantages, RDF uses graph structures instead of trees.

RDF Prefers Graphs Over Tress

- RDF was designed to describe general relationships between resources, not to structure documents.
- It focuses on expressing how different objects (resources) are related, rather than organizing them in a hierarchy.
- Multiple relationships between resources naturally form a graph structure, making graphs the preferred representation in RDF.

RDF Prefers Graphs Over Tress

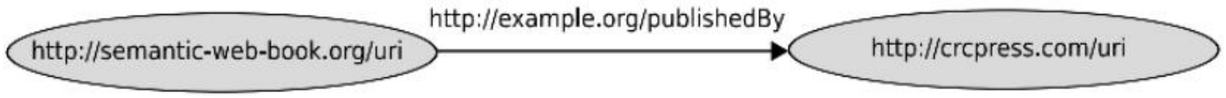
- Also, RDF was designed to describe data on the WWW and electronic networks.
- Information on the web is typically decentralized and stored in various locations.
- RDF graphs can easily be combined from multiple sources, forming a larger graph.
- In contrast to trees, graphs are more suitable for integrating distributed and related data due to their flexible structure.

Names in RDF: URIs

- RDF graphs support easy composition of distributed data structures.
- However, composing intended information across graphs presents challenges.
- A key issue is the inconsistency of resource identifiers across different RDF documents.
  - Same resource has different identifiers.
  - Same identifier refers to different resources.

Names in RDF: URIs

- **Example**: A simple RDF graph describing the relationship between this book and the publisher, CRC Press.



- Same resource has different identifiers (e.g., multiple URIs for the same book).
- Same identifier refers to different resources (e.g., “CRC” could mean different things).

Names in RDF: URIs

- This identifier ambiguity makes automatic processing and integration of RDF data problematic.
- As a solution, RDF uses **URIs (Uniform Resource Identifiers)** to uniquely identify resources.
- URIs are a broader concept than **URLs.**
- Every URL is a valid URI, and URLs can be used in RDF to identify web resources.

URIs and URNs

- RDF is often used to represent information not just about web pages, but about diverse types of objects (e.g., books, people, places).
- These objects may not be accessible online, so their **URIs serve purely for identification**, not for retrieval.
- **URIs that are not URLs** (i.e., not web-accessible) are often referred to as **URNs (Uniform Resource Names)**.

URI Construction Scheme

- Construction of URIs follows a similar scheme to Web addresses.
- **General format of a URI is:**

```
scheme:[//authority] path [?query] [#fragment]
```

- **Scheme:**
  - Identifies the type or classification of the URI.
  - Examples: http, ftp, mailto, file, irc.

URI Construction Scheme

- **Authority:**
  - Refers to a domain or authority that further structures the URI, including user info & port details.
  - Preceded by //, and used in web-related URIs.
  - Examples: `semantic-web-book.org`, `john@example.com`, `example.org:8080`.
- **Path:**
  - Represents the main resource or identifier, usually hierarchical.
  - Can be empty or have a structure with / separators.
  - Examples: `/etc/passwd`, `/this/path/with/-:_~.`

URI Construction Scheme

- **Query:**
  - Optional component used to pass additional information, often for web services or searches.
  - Recognized by the ? symbol.
  - Example: `q=Semantic+Web+book.`
- **Fragment:**
  - Optional part that identifies a sub-resource, such as a section within a document.
  - Recognized by the # symbol.
  - Example: `section`.

URI Construction Scheme

- Example:

  `http://semantic-web-book.org/uri`

- This URI may refer to a book without implying the presence of a retrievable document at that location.
- A URI's most important component is the scheme.
- Schemes like "http" are often linked with protocols for data transmission but are used in URIs that don’t necessarily point to a Web location.

Names in RDF: URIs

- RDF graphs use URIs to label nodes and edges, distinguishing them from other resources.

- Exceptions include:
  - **Data values** that are not URIs.
  - **Blank nodes** which lack names.

Data Values in RDF: Literals

- URIs are used to name abstract resources, even those that cannot be directly represented or processed by a computer.
- URIs act as **references** to resources like people, books, publishers, etc.
- Interpretation of URIs is not formally defined; different tools or services may interpret them in their own way.

Data Values in RDF: Literals

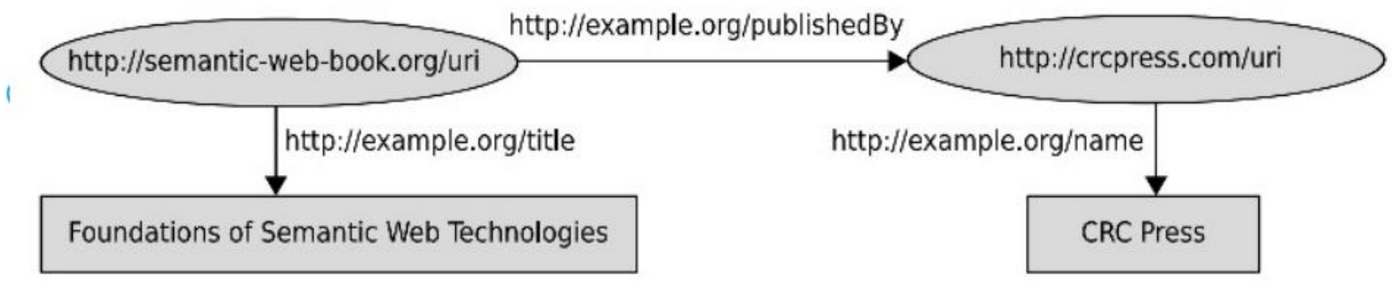
- For instance, a web service may recognize URIs referring to books and treat them by showing purchasing options or prices.
- This flexibility is useful when dealing with resources, but the interpretation of certain values can be specific to the application.
- Concrete data values, such as numbers , times, or truth values have consistent interpretations across all contexts.
- E.g. number 42 always has the same meaning.

Data Values in RDF: Literals

- Data values in RDF are represented by **literals**, which are reserved names for RDF resources with a specific datatype.
- Value of a literal is usually described by a sequence of characters (e.g., "42").
- Interpretation of the sequence depends on its **datatype**.
- For example, "42" and "042" represent the same number but are different text strings.

Data Values in RDF: Literals

- Literals without a specified datatype: **Untyped literals** are considered text strings by default.
- As can be seen in Figure, rectangular boxes are used to distinguish literals.



Data Values in RDF: Literals

- Literals cannot be the origin of edges in RDF graphs, meaning direct statements about literals are not allowed.
- Literals cannot be used as labels for edges in RDF graphs, but they can be used for nodes.
- In RDF, there is no principle separation between resources used for labeling nodes and edges.
- Literals without a datatype are interpreted as text strings.

Syntax for RDF

- So far, RDF graphs were illustrated using diagrams.
- Diagrams are easy to read and precise but unsuitable for computer processing.
- Visual graph representation is only effective for very small graphs.
- Large datasets (with thousands or millions of nodes) cannot be practically stored or shared as images.

Syntax for RDF

- A textual representation of RDF is needed for electronic storage and processing.
- This involves breaking down the RDF graph into smaller, manageable parts.
- Converting complex data structures into linear character strings is known as **serialization**.

From Graphs to Triples

- Graphs in computer science can be represented in various ways, such as adjacency matrices.
- RDF (Resource Description Framework) graphs are typically sparse, meaning most possible relationships do not exist.
- Instead of representing all possible edges, RDF graphs are stored as a set of actual edges only.

From Graphs to Triples

- Each edge is stored individually & consists of:
  - **Subject** (e.g., start point URI)
  - **Predicate** (e.g., relationship label)
  - **Object** (e.g., endpoint URI)
- Eg:

Subject - http://semantic-web-book.org/uri
Predicate - http://example.org/publishedBy
Object - http://crcpress.com/uri

From Graphs to Triples

- RDF graphs are essentially fully described by their edges.
- Each edge corresponds to an **RDF triple**: **subject–predicate–object**.
- Each part of a triple can be a URI; the object can also be a literal.
- RDF graphs are typically represented as a collection of triples, listed in any order.

Triple Syntax: N3, N-Triples

- The basic idea of Triples was taken up in various concrete proposals for serializing RDF.
- Tim Berners-Lee introduced **Notation 3 (N3)** in 1998, extending RDF with paths, rules, and more complex expressions.
- In 2004, the RDF recommendation proposed a simpler subset of N3 called **N-Triples** as an RDF syntax.

Turtle Syntax

- N-Triples was later extended with abbreviations and improvements, resulting in RDF **Turtle** syntax.
- Both **N-Triples** and **Turtle** are subsets of **N3**, limited to expressing valid RDF graphs.
- **Turtle (Terse RDF Triple Language)** was formalized in an official standardization document by W3C in 2014.

https://www.w3.org/TR/turtle/

Turtle Syntax

```
<http://semantic-web-book.org/uri>
    <http://example.org/publishedBy>  <http://crcpress.com/uri> .
<http://semantic-web-book.org/uri>
    <http://example.org/title>
        "Foundations of Semantic Web Technologies" .
<http://crcpress.com/uri>
    <http://example.org/name>         "CRC Press" .
```

- **URIs** are enclosed in **angular brackets** (< >).
- **Literals** are enclosed in **quotation marks**.
- Every **statement ends with a full stop** (.).

Turtle Syntax

- Besides above characteristics, the syntax is a direct translation of the RDF graph into triples.
- **Spaces and line breaks** matter only **within URIs or literals** — otherwise, they're ignored.
- However long names often require splitting a single triple over multiple lines.
- Due to the hierarchical structure of URIs, the identifiers in RDF documents typically use similar prefixes.

Turtle Syntax

- Turtle offers a mechanism for abbreviating such URIs using so-called **namespaces**.

```
@prefix book: <http://semantic-web-book.org/> .
@prefix ex: <http://example.org/> .
@prefix crc: <http://crcpress.com/> .

book:uri  ex:publishedBy  crc:uri .
book:uri  ex:title        "Foundations of Semantic Web Technologies" .
crc:uri   ex:name         "CRC Press" .
```

- URIs are now abbreviated using prefixes of the form **`prefix:`**.

Turtle Syntax

- Angular brackets (<>) are no longer used to enclose abbreviated URIs.
- Without the above modification it would be possible to confuse the abbreviated forms with full URIs.
- Prefixes text can be freely chosen, but it's recommended to use clear, readable abbreviations that hint at what they represent.
- Identifiers in the form **`prefix:name`** are called **QNames** (Qualified Names).

Turtle Syntax

- Often, RDF descriptions have many triples with the same subject, or the same subject and predicate.
- Turtle syntax offers shortcuts for these common cases.

```
@prefix book: <http://semantic-web-book.org/> .
@prefix ex: <http://example.org/> .
@prefix crc: <http://crcpress.com/> .

book:uri  ex:publishedBy  crc:uri ;
          ex:title        "Foundations of Semantic Web Technologies" .
crc:uri   ex:name         "CRC Press", "CRC" .
```

Turtle Syntax

- A **semicolon (;)** ends a triple and **keeps the same subject** for the next triple.
- A **comma (,)** ends a triple but **keeps both the subject and predicate** for the next triple.
- This makes it possible to write multiple triples without repeating the subject or predicate.
- Above example discussed results in four RDF triples involving four nodes and four edges.

Turtle Syntax

- Further the Semicolons and commas can be combined to simplify multiple triples.

```
@prefix book: <http://semantic-web-book.org/> .
@prefix ex: <http://example.org/> .

book:uri  ex:author book:Hitzler, book:Krötzsch, book:Rudolph ;
          ex:title  "Foundations of Semantic Web Technologies" .
```

Turtle Syntax

- Turtle syntax is a machine-processable and relatively human-readable RDF representation.
- Turtle is not the most widely used RDF syntax in practice.
- There is a limited support in programming languages.
- Many languages lack standard libraries for Turtle, requiring custom tools for handling it.

XML Serialization of RDF

- In contrast, essentially every programming language offers libraries for processing XML files.
- Therefore, the main syntax for RDF is the XML-based serialization **RDF/XML,** building on existing XML solutions for storage and pre-processing.
- It offers extra features and abbreviations for representing advanced concepts.

RDF/XML

- XML uses a tree (hierarchical) data model, while RDF uses a graph data model.
- RDF/XML must follow XML's hierarchical structure.
- RDF triples need to be encoded hierarchically due to XML's structure.
- Turtle syntax is more space-efficient and is often useful to assign multiple predicate-object pairs to a single subject.

RDF/XML

- Triples in RDF/XML are **grouped by their subjects**.

```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:ex ="http://example.org/">

  <rdf:Description rdf:about="http://semantic-web-book.org/uri">
    <ex:publishedBy>
      <rdf:Description rdf:about="http://crcpress.com/uri">
      </rdf:Description>
    </ex:publishedBy>
  </rdf:Description>

</rdf:RDF>
```

RDF/XML

- An RDF/XML document may optionally begin with a specification of XML version and encoding.
- Document starts with a root element **`<rdf:RDF>`**.
- Global XML namespaces (like **`ex:`** and **`rdf:`**) are declared within this root element.
- Similar to Turtle, namespaces abbreviate URIs using QNames, based on the XML namespace mechanism.

RDF/XML

- Although namespace prefixes are mostly arbitrary, it's conventional to use `rdf:` for the RDF namespace.
- Elements with special meaning in RDF/XML are recognized by the `rdf:` prefix.
- The `rdf:RDF` element contains the encoding of RDF triples.
- **Subjects and objects** are described using `rdf:Description` elements, with the `rdf:about` attribute specifying the resource's identifier.

RDF/XML

- **Predicate** is represented directly as an element (e.g., `ex:publishedBy`).
- Multiple triples are encoded with separate `rdf:Description` elements — multiple descriptions may refer to the same subject.
- Order of triples is irrelevant in RDF encoding.
- Nested `rdf:Description` elements can be used to create a more concise serialization.

RDF/XML

```
<rdf:Description rdf:about="http://semantic-web-book.org/uri">
  <ex:title>Foundations of Semantic Web Technologies</ex:title>
  <ex:publishedBy>
    <rdf:Description rdf:about="http://crcpress.com/uri">
      <ex:name>CRC Press</ex:name>
    </rdf:Description>
  </ex:publishedBy>
</rdf:Description>
```

- Literals are represented directly within the contents of a predicate element.
- For example, the name "`CRC Press`" is nested in XML elements instead of using a separate subject element for `http://crcpress.com/uri`.

RDF/XML

- Some further abbreviations are possible.

```
<rdf:Description rdf:about="http://semantic-web-book/uri"
                 ex:title= "Foundations of Semantic Web Technologies">
  <ex:publishedBy rdf:resource="http://crcpress.com/uri" />
</rdf:Description>
<rdf:Description rdf:about="http://crcpress.com/uri"
                                                ex:Name="CRC Press" />
```

- Predicates with **literal objects** are encoded as **XML attributes**.
- This encoding method is only valid for **literals**, not **URIs** as they can be be misinterpreted as literal strings.

Datatypes in RDF

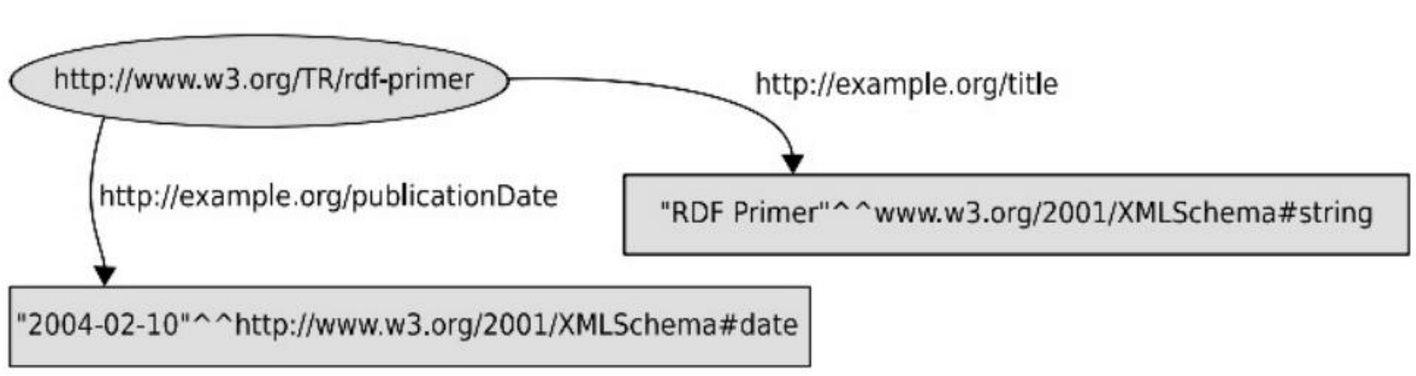
- RDF supports data values through **literals**.
- So far, literals were treated only as **character strings**.
- Practical applications need more complex **datatypes** (e.g., numbers, dates).
- **Datatypes** influence how values are interpreted.

Datatypes in RDF

- Example: Sorting the values “10”, “02”, “2”:
  - As strings: Alphabetical order → “02” < “10” < “2”.
  - As numbers: Numeric order → “2” = “02” < “10”.
- RDF allows literals to have explicit datatypes.
- Each datatype is uniquely identified by a URI.
- While any URI could technically be used, widely supported datatype URIs are preferred.
- RDF recommends using **XML Schema** datatypes for broader compatibility.

Datatypes in RDF

- Figure illustrates an RDF graph can include **datatype information** alongside literal values.



- RDF Primer document has a title (string) and publication date (date).
- Literal values are represented as: `"value"^^<datatype URI>.`

Datatypes in RDF

- Typed literals in RDF are treated as single elements and they behave similarly to untyped literals.
- In Turtle syntax, datatype URIs can be abbreviated using namespaces.
- If written as full URIs in Turtle, they must be enclosed in angular brackets (<>).

```
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
<http://www.w3.org/TR/rdf-primer>
       <http://example.org/title>  "RDF Primer"^^xsd:string ;
       <http://example.org/publicationDate> "2004-02-10"^^xsd:date .
```

Datatypes in RDF

- RDF/XML uses a different representation by including the **`rdf:datatype`** attribute.

```
<rdf:Description rdf:about="http://www.w3.org/TR/rdf-primer">
  <ex:title rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
    RDF Primer
  </ex:title>
  <ex:publicationDate
              rdf:datatype="http://www.w3.org/2001/XMLSchema#date">
    2004-02-10
  </ex:publicationDate>
</rdf:Description>
```

Language Settings & Datatypes

- Untyped literals in RDF have **no assigned type**, though they often behave like `xsd:string` in practice.
- A key difference between typed and untyped literals arises with **language information**.
- XML allows the specification of natural language for content using the `xml:lang` attribute.

```
xml:lang="en" or xml:lang="de-ch"
```

Language Settings & Datatypes

- Language information in XML is **hierarchical**: child elements inherit the language of their parent unless overridden.
- Language information can also be provided in RDF/XML, but this is semantically relevant only for untyped literals.

```
<rdf:Description rdf:about="http://www.w3.org/TR/rdf-primer">
   <ex:title xml:lang="fr">Initiation à RDF</ex:title>
   <ex:title xml:lang="en">RDF Primer</ex:title>
</rdf:Description>
```

Language Settings & Datatypes

- In Turtle, language information is supplied by means of the symbols @.

```
<http://www.w3.org/TR/rdf-primer> <http://example.org/title>
   "Initiation à RDF"@fr, "RDF Primer"@en .
```

- Language settings in RDF are part of the data value.

Many-Valued Relationships

- So far, only simple binary relationships between resources have been represented.
- These binary relationships form a **directed graph** structure.
- A question arises: Can this simple graph structure represent more complex data?
- How RDF can indeed encode relationships involving more than two resources?

Many-Valued Relationships

- Following excerpt from an RDF description formalizes ingredients of a cooking recipe.

```
@prefix ex: <http://example.org/> .
ex:Chutney  ex:hasIngredient   "1lb green mango",
                               "1tsp. Cayenne pepper" .
```

- Initial encoding models ingredients and their amounts as plain text strings.
- This method is unsatisfactory because it limits the ability to query specific ingredients without parsing entire strings.

Many-Valued Relationships

- A better approach would:
  - Represent ingredients and their amounts as separate resources.
  - Enable more precise and flexible querying.

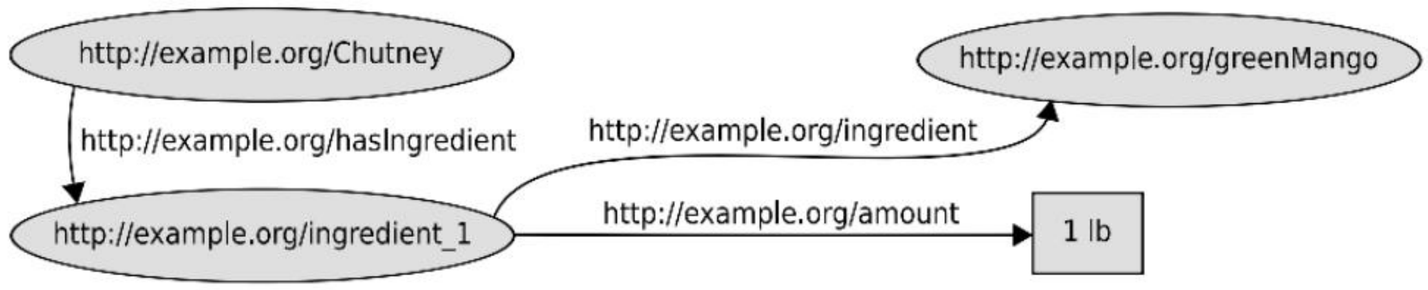
```
@prefix ex: <http://example.org/> .
ex:Chutney  ex:ingredient   ex:greenMango;     ex:amount   "1lb" ;
            ex:ingredient   ex:CayennePepper;  ex:amount   "1tsp." .
```

- However, this approach also fails due to limitations in expressing complete relationships

Many-Valued Relationships

- The example discusses a **three-valued (ternary) relationship** between a recipe, an ingredient, and an amount.
- RDF cannot directly represent such **ternary** or **n-ary** relationships.
- To handle this, **auxiliary nodes** are introduced into the RDF graph.
- These auxiliary nodes serve as **connectors** between the subject (e.g., a recipe) and multiple related objects (e.g., ingredient and amount).

Many-Valued Relationships



- Here the node `ex:Ingredient1` plays the role of an explicit connection between recipe, ingredient, and amount.
- Additional auxiliary nodes are created for each ingredient, linking all relevant components.

Many-Valued Relationships

- This method enables the connection of **multiple values** to a single subject.
- However, it requires **extra URIs**:
  - One for the auxiliary node itself.
  - Additional triples with new predicates (e.g., `ex:hasIngredient and ex:ingredient`).
- Naming of predicates can reflect the importance of certain values.

Many-Valued Relationships

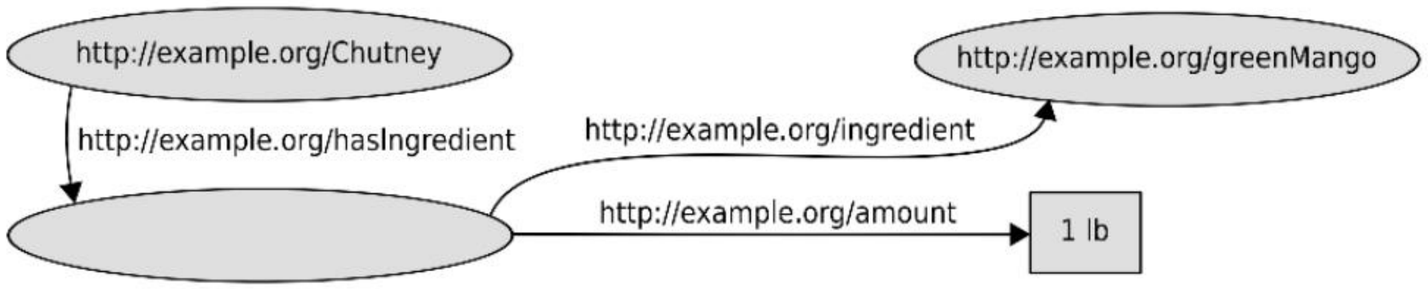
- RDF provides a reserved predicate, **rdf:value**, to indicate the main object in a many-valued relation.

```
@prefix ex:  <http://example.org/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
ex:Chutney        ex:hasIngredient  ex:ingredient1 .
ex:ingredient1    rdf:value    ex:greenMango;
                  ex:amount    "1lb" .
```

Blank Nodes

- Modeling many-valued relationships in RDF may require auxiliary (helper) nodes.
- These auxiliary nodes usually do not represent explicitly described resources.
- Such nodes serve a structural role and often don’t need a globally referable URI.
- RDF provides **blank nodes** (or **bnodes**) for this purpose.

Blank Nodes



- Blank nodes have no URI and are used to represent unnamed resources.
- They are typically used as subjects or objects in RDF triples (not predicates).
- Predicates (edges) **must** always have a URI.

Blank Nodes

- Blank nodes cannot be addressed globally by means of URIs, and they do not carry any additional information within RDF graphs.
- However, **serialization of RDF** requires a way to refer to specific blank nodes within a document.
- To handle this, blank nodes are assigned **IDs** within a document context.
- In RDF/XML, this is done using the `rdf:nodeID` attribute.

Blank Nodes

```
<rdf:Description rdf:about="http://example.org/Chutney">
   <ex:hasIngredient rdf:nodeID="id1" />
</rdf:Description>
<rdf:Description rdf:nodeID="id1">
   <ex:ingredient rdf:resource="http://example.org/greenMango" />
   <ex:amount>1lb</ex:amount>
</rdf:Description>
```

- The label id1 is only relevant within the specific RDF document it appears in.
- Node IDs are a syntactic mechanism used to serialize blank nodes.

11/4/25